# Project Documentation – Team SLAK

Shao Gu (Shawn) Lu, Liufei Chen, Zhixuan (Alex) Jiang, Kanika Selvapandian

---

**Abstract** **2**
**Background and Motivation** **3**
**Methodology** **3**
**Code Documentation** **5**
Functionality 5
Functions 6
Functions in compiled_data.py 6
User Instructions 14
Environment Setup 14
Required Libraries 14
File Requirements 15
Steps to Run the Program 15
**Reference** **16**

# Abstract

Analyzed voter turnout in Pennsylvania counties through Python-based linear and panel regression models. The project involved data preprocessing using pandas and NumPy, as well as building predictive models with scikit-learn. Visualizations created with Matplotlib and plotly highlighted key trends and relationships. The study integrated socio-economic and demographic data to identify variables that influence voter turnout over time across different regions.

# Background and Motivation

A free and fair election is one of the conditions to be a democracy (Huntington, 1991), and citizens' level of participation in elections is a key indicator of the quality of a democracy (Powell, 1982). Unfortunately, the voter turnout rates in the United States have been lower than the other democracies for a long time. This situation is even worse in two decades.

Why are American people reluctant to vote? Political scientists have dedicated themselves to explaining this phenomenon since as early as the 1960s. Downs (1957) and Riker and Ordeshook (1968) approach individuals' voting behaviors with a rational choice perspective. They believe citizens, as rational men, will calculate the benefits and costs to decide whether they want to vote. On the other hand, Almond and Verba (1963) provide a cultural alternative, that the political culture within a society drives their members' political participation. These two theories, rational choice, and political culture theory are the foundations for the various theories proposed by the later researchers. Jackman (1987) and Blais (2006) argue that different institutions, such as electoral systems and political systems, provide different incentives for citizens to vote. Individual's or Nation-wide Economic conditions (Ojeda, 2018; Kasara & Suryanarayan, 2015) and citizen's knowledge and education level ( Sondheimer & Green, 2010; Lassen, 2005) are also influential to individuals' voting turnout. Additionally, Nagel & McNulty (1996) and Fraga et al. (2022) point out that partisanship is crucial for individuals' voting behavior.

There is more interesting research that we cannot list all of them, but the diverse perspectives on voter turnout make us curious about which factors are more important than the others. As a result, this project aims to provide a whole picture of which factors do affect real-world voter turnout, while others do not.

# Methodology

The study uses actual voter turnout rates and social economic, and demographic variables at the county level in Pennsylvania for years 2016 and 2020. This strategy can avoid the common problems from the self-reported survey data, such as self-reporting errors, social desirability bias,

selection bias, and challenges with population representativeness. However, it should be noted that our project does not provide any assumption or theory to explain voter turnout. We include as many variables, which are indicated in journal articles, as we can in our regression analysis. We also adopted the Ordinal Linear Regression model and the Panel Regression Model to identify which variables are statistically significant and compare the performance of different models.

Our data is at the county level. For our dependent variable, voter turnout rate, we collect data from the 2016 and 2020 presidential elections for each county. We, accordingly, collect the socio-economic and demographic data for 2016 and 2020, respectively. Our operationalizations for each variable are presented in Table 1.

**Table 1: the Operationalization of Variables**

| **Variables** | **Operationalization** |
|---|---|
| Voter Turnout | $\frac{Total\ Votes}{Voting-Eligible\ Population}$ |
| Gender | $\frac{Female\ Population}{Total\ Population}$ |
| Education | $\frac{Population\ with\ College\ Degree\ or\ Above}{Voting-Eligible\ Population}$ |
| Ethnic Groups | |
| Black | Black Population / Total Population |
| Hispanic | Hispanic Population / Total Population |
| Asian | Asian Population / Total Population |
| Income | The Median Annual Income |
| Employment | The Unemployment Rate |
| Urban/Rural | $\frac{Total\ Population}{Land\ Area}$ |
| Partisanship | The Democrat Presidential Candidate's Vote Share at Last Presidential Election |

# Code Documentation

## *Functionality*

1. **compiled_data.py**: Collects data from web scraping and APIs, aggregates it with CSV files downloaded from the web, and performs data cleaning to produce a cleaned data CSV file.
2. **regression.py:** Doing regression analysis by using the OLS method, to find out which variables are correlated with turnout rate with a high significance level.
3. **p_panel.py:** Using panel regression method to identify which variables have a significant correlation with voter turnout rates. Additionally, we are comparing two different regression methods to determine which offers a better fit for the data.
4. **visualization:** Visualizing the data by plotting Voter Turnout Increase/Decrease for All Counties, Voter Turnout Rate by County and Urban/Rural Classification, Democratic Votes Distribution Map (2016), Democratic Votes Distribution Map (2020), Racial Composition and Voter Turnout Comparison (2016 and 2020), Aggregated County Correlation Heatmap.

# *Functions*

## Functions in compiled_data.py

### *Education Data*

```
"""
Extract the county name from the column header.
For example: if the column header is "Adams County, Pennsylvania!!Percent!!Estimate",
this function extracts "Adams" from the column header.
"""
def extract_county_name(full_name):
    return full_name.split(',')[0].replace('County', '').strip()
```

```
"""
Convert a percentage string to a float (e.g., "40.0%" becomes 40.0).
Returns None if conversion fails (e.g., "N/A").

Used to process education percentages for ages 18-24 and 25+,
enabling calculations of combined education levels across age groups.
"""
def parse_percentage(value):
    try:
        return float(value.strip('%'))
    except ValueError:
        return None
```

### *Unemployment Data*

```
"""
Extracts unemployed data for counties from the specified row of a DataFrame.
Parameters:
- data: DataFrame containing unemployment data.
- index_row: The row index that holds unemployment percentage data.
- year: The year of the data (e.g., 2016 or 2020).
"""
def extract_unemployment(data, index_row, year):
    percent_columns = [col for col in data.columns if 'County' in col and '!!Percent' in col]
    unemployment_data = data.loc[index_row, percent_columns]
    unemployment_data.index = [col.split('County')[0].strip() for col in percent_columns]
    unemployment_data_df = unemployment_data.reset_index().rename(columns={'index': 'County', index_row: 'Unemployment Rate'})
    unemployment_data_df['Year'] = year
    unemployment_data_df = unemployment_data_df[~unemployment_data_df['Unemployment Rate'].str.contains('±')]
    return unemployment_data_df
```

### *Income Data*

```
"""
Converts a percentage string to a float rounded to one decimal place.
"""
def parse_percentage(value):
    try:
        return round(float(value.strip('%')), 1)
    except (ValueError, AttributeError):
        return None
```

```
"""
Extracts the percentage of individuals with income less than $50,000 for each county.

Parameters:
- df: DataFrame containing income data.
- year: The year of the data (e.g., 2016 or 2020).
"""
def extract_low_income_data(df, year):
    county_columns = [col for col in df.columns if 'County, Pennsylvania!!Percent' in col]
    county_data = {'County': [], 'Year': [], 'Income < 50000': []}

    for county_column in county_columns:
        low_income_sum = 0
        for row in low_income_rows:
            try:
                value = df.iloc[row, df.columns.get_loc(county_column)]
                if pd.notnull(value):
                    parsed_value = parse_percentage(value)
                    if parsed_value is not None:
                        low_income_sum += parsed_value
            except IndexError:
                continue
        county_name = county_column.split(',')[0].replace('County', '').strip()
        if low_income_sum > 0:
            county_data['County'].append(county_name)
            county_data['Year'].append(year)
            county_data['Income < 50000'].append(f"{low_income_sum:.1f}%")
    return pd.DataFrame(county_data).set_index(['County', 'Year'])
```

### *Sex Data*

```
"""
Extracts the male percentage data for each county.

Parameters:
- data: DataFrame containing sex data.
- index_row: The row index that holds the male percentage data.
- year: The year of the data (e.g., 2016 or 2020).
"""
def extract_male(data, index_row, year):
    percent_columns = [col for col in data.columns if 'County' in col and '!!Percent' in col]
    male2016_data = data.loc[index_row, percent_columns]
    male2016_data.index = [col.split('County')[0].strip() for col in percent_columns]
    male2016_data_df = male2016_data.reset_index().rename(columns={'index': 'County', index_row: 'Male Percentage'})
    male2016_data_df['Year'] = year
    male2016_data_df = male2016_data_df[~male2016_data_df['Male Percentage'].str.contains('±')]
    return male2016_data_df
```

```
"""
Extracts the female percentage data for each county.

Parameters:
- data: DataFrame containing sex data.
- index_row: The row index that holds the female percentage data.
- year: The year of the data (e.g., 2016 or 2020).
"""
def extract_female(data, index_row, year):
    percent_columns = [col for col in data.columns if 'County' in col and '!!Percent' in col]
    female2016_data = data.loc[index_row, percent_columns]
    female2016_data.index = [col.split('County')[0].strip() for col in percent_columns]
    female2016_data_df = female2016_data.reset_index().rename(columns={'index': 'County', index_row: 'Female Percentage'})
    female2016_data_df['Year'] = year
    female2016_data_df = female2016_data_df[~female2016_data_df['Female Percentage'].str.contains('±')]
    return female2016_data_df
```

## *Race Data*

```
"""
Extracts the Hispanic percentage data for each county.

Parameters:
- data: DataFrame containing race data.
- index_row: The row index that holds the Hispanic percentage data.
- year: The year of the data (e.g., 2016 or 2020).
"""
def extract_hispanic(data, index_row, year):
    percent_columns = [col for col in data.columns if 'County' in col and '!!Percent' in col]
    hispanic2016_data = data.loc[index_row, percent_columns]
    hispanic2016_data.index = [col.split('County')[0].strip() for col in percent_columns]
    hispanic2016_data_df = hispanic2016_data.reset_index().rename(columns={'index': 'County', index_row: 'Hispanic percentage'})
    hispanic2016_data_df['Year'] = year
    hispanic2016_data_df = hispanic2016_data_df[hispanic2016_data_df['Hispanic percentage'].apply(lambda x: x.replace('%', '').replace('.',
    return hispanic2016_data_df
```

```
"""
Extracts the Black percentage data for each county.

Parameters:
- data: DataFrame containing race data.
- index_row: The row index that holds the Black percentage data.
- year: The year of the data (e.g., 2016 or 2020).
"""
def extract_black(data, index_row, year):
    percent_columns = [col for col in data.columns if 'County' in col and '!!Percent' in col]
    black2016_data = data.loc[index_row, percent_columns]
    black2016_data.index = [col.split('County')[0].strip() for col in percent_columns]
    black2016_data_df = black2016_data.reset_index().rename(columns={'index': 'County', index_row: 'Black percentage
    black2016_data_df['Year'] = year
    black2016_data_df = black2016_data_df[black2016_data_df['Black percentage'].apply(lambda x: x.replace('%', '').r
    return black2016_data_df
```

```
"""
Extracts the Native American percentage data for each county.

Parameters:
- data: DataFrame containing race data.
- index_row: The row index that holds the Native American percentage data.
- year: The year of the data (e.g., 2016 or 2020).
"""
def extract_nativeamerican(data, index_row, year):
    percent_columns = [col for col in data.columns if 'County' in col and '!!Percent' in col]
    nativeamerican2016_data = data.loc[index_row, percent_columns]
    nativeamerican2016_data.index = [col.split('County')[0].strip() for col in percent_columns]
    nativeamerican2016_data_df = nativeamerican2016_data.reset_index().rename(columns={'index': 'County', index_row: 'Native American percentage'})
    nativeamerican2016_data_df['Year'] = year

    # Filter out rows where the 'nativeamerican percentage' contains non-numeric values
    nativeamerican2016_data_df = nativeamerican2016_data_df[nativeamerican2016_data_df['Native American percentage'].apply(lambda x: x.replace('%', '').

    return nativeamerican2016_data_df
```

```
"""
Extracts the Asian percentage data for each county.

Parameters:
- data: DataFrame containing race data.
- index_row: The row index that holds the Asian percentage data.
- year: The year of the data (e.g., 2016 or 2020).
"""
def extract_asian(data, index_row, year):
    percent_columns = [col for col in data.columns if 'County' in col and '!!Percent' in col]
    asian2016_data = data.loc[index_row, percent_columns]
    asian2016_data.index = [col.split('County')[0].strip() for col in percent_columns]
    asian2016_data_df = asian2016_data.reset_index().rename(columns={'index': 'County', index_row: 'Asian percentage'})
    asian2016_data_df['Year'] = year

    # Filter out rows where the 'asian percentage' contains non-numeric values
    asian2016_data_df = asian2016_data_df[asian2016_data_df['Asian percentage'].apply(lambda x: x.replace('%', '').replace(','

    return asian2016_data_df
```

```
"""
Extracts the Hawaiian percentage data for each county.

Parameters:
- data: DataFrame containing race data.
- index_row: The row index that holds the Hawaiian percentage data.
- year: The year of the data (e.g., 2016 or 2020).
"""
def extract_hawaiian(data, index_row, year):
    percent_columns = [col for col in data.columns if 'County' in col and '!!Percent' in col]
    hawaiian2016_data = data.loc[index_row, percent_columns]
    hawaiian2016_data.index = [col.split('County')[0].strip() for col in percent_columns]
    hawaiian2016_data_df = hawaiian2016_data.reset_index().rename(columns={'index': 'County', index_row: 'Hawaiian percentage'})
    hawaiian2016_data_df['Year'] = year

    # Filter out rows where the 'hawaiian percentage' contains non-numeric values
    hawaiian2016_data_df = hawaiian2016_data_df[hawaiian2016_data_df['Hawaiian percentage'].apply(lambda x: x.replace('%', '').rep

    return hawaiian2016_data_df
```

```
"""
Extracts the Other Races percentage data for each county.

Parameters:
- data: DataFrame containing race data.
- index_row: The row index that holds the Other Races percentage data.
- year: The year of the data (e.g., 2016 or 2020).
"""
def extract_otherRaces(data, index_row, year):
    percent_columns = [col for col in data.columns if 'County' in col and '!!Percent' in col]
    otherRaces2016_data = data.loc[index_row, percent_columns]
    otherRaces2016_data.index = [col.split('County')[0].strip() for col in percent_columns]
    otherRaces2016_data_df = otherRaces2016_data.reset_index().rename(columns={'index': 'County', index_row: 'Other Races percentage'})
    otherRaces2016_data_df['Year'] = year

    # Filter out rows where the 'otherRaces percentage' contains non-numeric values
    otherRaces2016_data_df = otherRaces2016_data_df[otherRaces2016_data_df['Other Races percentage'].apply(lambda x: x.replace('%', '').r

    return otherRaces2016_data_df
```

```
"""
Extracts the More Than One Race percentage data for each county.

Parameters:
- data: DataFrame containing race data.
- index_row: The row index that holds the More Than One Race percentage data.
- year: The year of the data (e.g., 2016 or 2020).
"""
def extract_more_than_oneRaces(data, index_row, year):
    percent_columns = [col for col in data.columns if 'County' in col and '!!Percent' in col]
    more_than_oneRaces2016_data = data.loc[index_row, percent_columns]
    more_than_oneRaces2016_data.index = [col.split('County')[0].strip() for col in percent_columns]
    more_than_oneRaces2016_data_df = more_than_oneRaces2016_data.reset_index().rename(columns={'index': 'County', index_row: 'More Than Or
    more_than_oneRaces2016_data_df['Year'] = year

    # Filter out rows where the 'more_than_oneRaces percentage' contains non-numeric values
    more_than_oneRaces2016_data_df = more_than_oneRaces2016_data_df[more_than_oneRaces2016_data_df['More Than One Races percentage'].apply

    return more_than_oneRaces2016_data_df
```

### *Total Population Data*

```
"""
Extracts the total population estimates for each county.

Parameters:
- data: DataFrame containing total population data.
- index_row: The row index that holds the total population estimate.
- year: The year of the data (e.g., 2016 or 2020).
"""
def extract_pop(data, index_row, year):
    percent_columns = [col for col in data.columns if 'County' in col and '!!Estimate' in col]
    pop2016_data = data.loc[index_row, percent_columns]
    pop2016_data.index = [col.split('County')[0].strip() for col in percent_columns]
    pop2016_data_df = pop2016_data.reset_index().rename(columns={'index': 'County', index_row: 'Total Pop percentage'})
    pop2016_data_df['Year'] = year
    return pop2016_data_df
```

### *Final Clean Data*

```
"""
# These functions clean and fill data in a DataFrame:
"""
def fill_null(name):
    for idx, item in enumerate(data[name]):
        if type(item) == float:
            if idx > 0:
                if data.loc[idx - 1, 'County'] == data.loc[idx, 'County']:
                    data.loc[idx, name] = data.loc[idx - 1, name]
                elif data.loc[idx + 1, 'County'] == data.loc[idx, 'County']:
                    data.loc[idx, name] = data.loc[idx + 1, name]

def clean_percentage(name):
    for idx, item in enumerate(data[name]):
        if type(item) == float:
            print(data.loc[idx, 'County'], item, name)
        else:
            data.loc[idx, name] = float(item.split('%')[0]) / 100

def for_sex(name):
    for idx, item in enumerate(data[name]):
        data.loc[idx, name] = float(item) / 100
```

## Functions in regression.py

```
"""
Converts a specific year's data from a DataFrame column into a list of floats.

Parameters:
- name: The name of the column to extract data from.
- year: The specific year to filter the data.
"""
def to_list(name, year):
    temp = []
    for idx, item in enumerate(data[name]):
        if data.loc[idx, 'Year'] == year:
            temp.append(float(item))
    return temp
"""
Converts all data from a DataFrame column into a list of floats.

Parameters:
- name: The name of the column to extract data from.
"""
def to_list_all(name):
    temp = []
    for idx, item in enumerate(data[name]):
        temp.append(float(item))
    return temp
```

```
"""
Generates a list of lists fit for use as independent variables in sklearn models.

Parameters:
- xList: A list to store the generated lists of independent variables.
- yList: A list of dependent variable values (e.g., turnout data).
- variables: A list of lists or arrays containing the independent variables.
"""
def x_generator(xList, yList, variables):
    for idx in range(0, len(yList)):
        temp = []
        for v in variables:
            temp.append(v[idx])
        xList.append(temp)
    return xList
```

```
"""
Calculates p-values for model coefficients and their significance levels.

Parameters:
- X: The matrix of independent variables (features).
- Y: The dependent variable (target values).
- model: A fitted linear regression model.
"""
def p_calculator(X, Y, model):
    residuals = Y - model.predict(X)
    n, k = X.shape
    RSS = np.sum(residuals**2)
    variance = RSS / (n - k - 1)

    # Compute the covariance matrix of the coefficients
    X_with_intercept = np.column_stack([np.ones(n), X])
    cov_matrix = variance * np.linalg.inv(X_with_intercept.T @ X_with_intercept)

    # Calculate t-statistics (coefficients / standard error)
    standard_errors = np.sqrt(np.diag(cov_matrix))
    t_stats = model.coef_ / standard_errors[1:]

    # Calculate p-values
    p_values = 2 * (1 - stats.t.cdf(np.abs(t_stats), df=n - k - 1))

    # demonstrate the significant levels
    sigStar = []
    for item in p_values:
        if item < 0.001:
            sigStar.append('***')
        elif item < 0.01:
            sigStar.append('**')
        elif item < 0.05:
            sigStar.append('*')
        else:
            sigStar.append('')

    return p_values, sigStar
```

```
"""
Performs linear regression, and calculates statistical measures.

Parameters:
- IV: List or array of independent variables (features).
- DV: List or array of dependent variable values (target).
- IVname: List of names corresponding to the independent variables.
"""
def regressionSKL(IV, DV, IVname):
    X = np.array(IV)
    Y = np.array(DV)

    model = LinearRegression()
    model.fit(X, Y)
    Y_pred = model.predict(X)

    coefficients = model.coef_.tolist()
    intercept = model.intercept_.tolist()

    n, k = X.shape
    p_values, sigStar = p_calculator(X, Y, model)

    # calculate log-likelihood
    residuals = Y - Y_pred
    sigma_squared = np.var(residuals, ddof=1)
    log_likelihood = -n/2 * np.log(2 * np.pi * sigma_squared) - (1/(2 * sigma_squared)) * np.sum(residuals**2)
    likelihood = np.exp(log_likelihood)

    # Adjusted R-squared
    r2 = r2_score(Y, Y_pred)
    adj_r2 = 1 - (1 - r2) * (n - 1) / (n - k - 1)

    # calculate BIC
    bic = np.log(n) * k - 2 * np.log(likelihood)

    print(f"sample size: {n}")
    print(f"Intercept: {intercept:.4f}")
    print(f"{'Variable':<15}{'Coefficient':<15}{'P-value':<15}{'Significance':<10}")
    for idx, item in enumerate(coefficients):
        print(f"{IVname[idx]:<15}{item:<15.4f}{p_values[idx]:<15.4f}{sigStar[idx]:<10}")
    print(f"R2 = {r2_score(Y, Y_pred):.4f}")
    print(f"Adjusted R2 = {adj_r2:.4f}")
    print(f"Log-Likelihood: {log_likelihood:.4f}")
    print(f"BIC is: {bic:.4f}")
```

## *User Instructions*

### *Environment Setup*

To run this program, it is recommended to use either **Jupyter Notebook** or **Spyder** in an **Anaconda environment**. Anaconda provides a comprehensive package management system and comes pre-installed with many essential data science libraries.

### *Required Libraries*

The following Python libraries must be installed to ensure the program runs smoothly. You can install them via **pip** or through **Anaconda** (Table 2).

**Table 2: Python Libraries to Install**

| Module | Installation |
|---|---|
| pandas | `pip install pandas` |
| requests | `pip install requests` |
| BeautifulSoup (bs4) | `pip install beautifulsoup4` |
| re (Regular Expressions) | built-in Python module, no need for installation |
| NumPy | `pip install numpy` |
| matplotlib | `pip install matplotlib` |
| seaborn | `pip install seaborn` |
| plotly | `pip install plotly` |
| geopandas | `pip install geopandas` |
| os | built-in Python module, no need for installation |
| scikit-learn (sklearn) | `pip install scikit-learn` |
| scipy | `pip install scipy` |
| statsmodels | `pip install statsmodels` |
| linearmodels | `pip install linearmodels` |

### *File Requirements*

1. Unzip the project folder into your working directory. This will extract the necessary Python files and any supporting data.
2. The project folder contains:
    a. 1. compiled.py
    b. 2. visualisation.py
    c. 3. regression.py
    d. data folder with necessary data files
    e. images folder which will save images from running visualisation.py
3. Once the folder is unzipped, move on to the next steps.

### *Steps to Run the Program*

1. Run the Data Compilation Script (1. compiled.py): This script will gather and process the required data from the different sources
    a. `python "1. compiled.py"`
2. Run the Visualization Script (2. visualisation.py): This script will generate visualizations using matplotlib, seaborn, or plotly
    a. `python "2. visualization.py"`
3. Run the Regression Analysis Script (3. regression.py): This script will perform statistical analysis which is linear regression and other statistical methods to validate the model using scikit-learn and scipy.
    a. `python "3. regression.py"`
4. Run the Panel Regression Analysis Script (4. panel.py): This script will conduct a panel data regression analysis, using statistical libraries such as pandas, numpy, linearmodels, and statsmodels
    a. `python "4. panel.py"`

# Reference


Huntington, S. P., 1991. The Third Wave: Democratization in the Late Twentieth Century, Norman: University of Oklahoma Press.

Ojeda, C. (2018), The Two Income-Participation Gaps. American Journal of Political Science, 62: 813-829. https://doi.org/10.1111/ajps.12375

Kasara, K. and Suryanarayan, P. (2015), When Do the Rich Vote Less Than the Poor and Why? Explaining Turnout Inequality across the World. American Journal of Political Science, 59: 613-627. https://doi.org/10.1111/ajps.12134

Powell GB. 1982. Comparative Democracies: Participation, Stability and Violence Cambridge, MA: Harvard Univ. Press

Jackman, R. W. (1987). Political institutions and voter turnout in the industrial democracies. American Political Science Review, 81(2), 405-423.

Blais, A. (2006). What affects voter turnout?. Annu. Rev. Polit. Sci., 9(1), 111-125.

Riker, W. H., and Ordeshook, P. C. (1968). A theory of the calculus of voting. American Political Science Review, 62(1): 25–42. https://doi.org/10.2307/1953324

Downs (1957). An Economic Theory of Democracy. New York: Harper & Row.

Nagel, J. H., & McNulty, J. E. (1996). Partisan Effects of Voter Turnout in Senatorial and Gubernatorial Elections. American Political Science Review, 90(4), 780–793. doi:10.2307/2945842

Fraga, B.L., Moskowitz, D.J. & Schneer, B. (2022). Partisan Alignment Increases Voter Turnout: Evidence from Redistricting. Political Behavior 44, 1883–1910. https://doi.org/10.1007/s11109-021-09685-y

Sondheimer, R.M. and Green, D.P. (2010), Using Experiments to Estimate the Effects of Education on Voter Turnout. American Journal of Political Science, 54: 174-189. https://doi.org/10.1111/j.1540-5907.2009.00425.x

Lassen, D.D. (2005), The Effect of Information on Voter Turnout: Evidence from a Natural Experiment. American Journal of Political Science, 49: 103-118. https://doi.org/10.1111/j.0092-5853.2005.00113.x